510.84
Il6r
no.271

[redacted]

7200-15

Report No. 271

ILLIAC IV

QUARTERLY PROGRESS REPORT

January, February and March 1968

Contract No.
US AF 30(602)4144

ILLIAC IV Document No. 192

DEPARTMENT OF COMPUTER SCIENCE · UNIVERSITY OF ILLINOIS · URBANA, ILLINOIS

ILLIAC IV

QUARTERLY PROGRESS REPORT

January, February and March 1968

Contract No.
US AF 30(602)4144

Department of Computer Science
University of Illinois
Urbana, Illinois
61801

June 10, 1968

This work was supported in part by the Department of Computer Science, University of Illinois, Urbana, Illinois, and in part by the Advanced Research Projects Agency as administered by the Rome Air Development Center, under Contract No. US AF 30(602)4144.

510.84
Il6r
no. 271-278
cop. 2

TABLE OF CONTENTS

1. REPORT SUMMARY

Burroughs and TI have agreed on a solution to the ECL noise problem. The solution, as mentioned in the previous Quarterly Progress Report, is to separate the voltage supply in the ECL logic module. This separation is accomplished by increasing the number of pins on the MSI package to 80 and the DIL package to 16 pins.

Burroughs is investigating the problem of power distribution to the PE's. The major problem is the switching of amperes in nanoseconds as the PE's change state during a microsequence. Several approaches are being studied and will be reported in the next QPR.

The I/O system design is nearing completion. Minor modifications have been suggested by the University and will be implemented by Burroughs. The bit density on the large disk has been changed from 4000 bits per inch to 3000 bits per inch. To maintain the 500×10^6 bits per second transfer rate at the lower density additional read-heads will be used.

The Translator Writing System was used to produce a one-pass compiler (a subset of ALGOL). The generated compiler was then used to successfully parse source programs. Continued emphasis will be placed on recovery due to source statement errors. Final documentation on ISL (Illinois Semantic Language) is in progress and will be available during the next quarter.

The diagnostic group has formulated plans with Burroughs on the diagnostic programs necessary to debug the breadboard PE. The University diagnostic group will produce a set of preliminary diagnostic programs that will output test sequences in the assembly language of the PE exercisers. These programs will provide automatic checkout of 75% of the breadboard PE. Additional diagnostic programs will be written by the logic designers at Burroughs.

The ILLIAC IV Programming Manual was delivered by Burroughs and is being distributed by the University.

Additional material on ILLIAC IV is being prepared in preparation for users from other Universities and government sources.

2. HARDWARE

2.1 Diagnostics

2.1.1 Algorithms for Test Generation

The two efforts to develop methods of generating test paths and patterns continued during this period.

2.1.1.1 Path Generating Method

In the preceding quarter, a program had been written which generates a set of test paths from a description of a directed graph. The generated path set is a minimal set sufficient for failure detection and location. This program is now expanded to allow up to 1000 nodes and 1000 arcs in the original graph. The number of characters for a node name can be as many as twelve rather than six. These changes to the program made it possible to use it for the generation of PE test sequence by the PEX which has direct access to all control enables.

2.1.1.2 Combinational Tests

The algorithm for generating a set of input patterns to test combinational networks at the chip level was improved. There are three conditions combined to determine the set of external inputs for testing a chip in the network: (1) to let the chip input signals take each of all possible value combinations; (2) to open any pseudo-feedback loop caused by fault; and (3) to steer the chip outputs to some of the external output lines to observe the fault. The generated pattern can detect any logical faults in a chip or wiring.

2.1.2 Methods for Fault Location

Several techniques for the identification of detected faults have been proposed and defined during this quarter. Evaluation of the techniques are on the way, and the result will be incorporated in the diagnostic program system.

The first technique makes use of a matrix that specifies whether a fault in a particular component affects the result of a particular test. The comparison of the test result with each column of this matrix shows up the existing fault(s).

Because of the difficulty anticipated in treating multiple faults and faults of a type other than logical modes, a supplement to the matrix processing method was examined which commits some of the location procedure to the engineer. A printed dictionary will be available to him in which he can look for suspected components through the indexing by the test results. All necessary information on the mechanization of the logic and circuits will be included in the dictionary.

The third approach is to order the tests in a branching tree so that when the test sequence is terminated somewhere in the tree, the terminating test number tells the fault location. An algorithm to optimize the location capability of the test tree for multiple faults is being developed and will be programmed for path tests of the PE by the PEX.

In some cases, the propagation of fault through the network is limited by the logic of the network, and a simple procedure for fault location using the concept of "runs" of fault is efficient.

The fifth approach is to process the test results by an analysis program which evaluates the Boolean derivative of the failed output signal. The evaluation is made with respect to each relevant internal signal to determine a subset of those relevant internal faults which could affect the failed external output under the test inputs.

2.1.3 PE Diagnostics

2.1.3.1 Path Tests

Two sets of paths were obtained by the PGM program for the diagnostic programs of the PE. The first set was generated from a graph whose nodes correspond to the registers and arcs correspond to the transfer instructions. The set has 22 paths for on-line testing by the CU.

The second path set was generated from a more detailed PE design. Each node corresponds to a portion of a register or a combinational network while the arcs correspond to the control enables ("F-signals") of which any subset can be selected for activation by the content of a MicroSequence Register in the PE Exerciser. The generated path set will be modified after the finalization of "F-signals" and used in the diagnostic program of the PEX.

2.1.3.2 <u>Combinational Tests</u>

Input patterns for combinational tests have been dervied for the CSA, CPA, BSW, MDG, MSG, LOD, and ADA.

The total number of tests which must be specified on the PEX tape is about five thousand. A number of input patterns are generated within the PE by modification of the data from the PEX.

The microsequences in the PEX program are obtained from the above combinational logic blocks.

Burroughs was informed of those results and has been working on their application to the diagnostic program system.

2.1.4 <u>CU Diagnostics</u>

A trip was made in March to Pasadena, California, to discuss the diagnostic program of B6500 with Burroughs' diagnosticians. The methods for CU diagnostics will be discussed in the next quarter based on our algorithms and Pasadena's.

2.2 <u>Design Automation</u>

A Design Automation Project was initiated during this quarter. The immediate goal will be to investigate the application of parallel organization to Design Automation. Within this effort, a study of ILLIAC IV's application to Logic Simulation via Ulrich's[6] proposed model and the technique used by Reiss[7] will be started. As a first step, the model will be developed and programmed on the B5500 and used to simulate logic and wiring delay on the ILLIAC IV logic board at Paoli.

NOTE: For pertinent papers on some of the subjects in this area, see References 1-5 which are listed at the end of this paper.

3. SOFTWARE

3.1 Translator Writing System

3.1.1 Introduction

Major progress towards the completion of the TWS project was accomplished during the quarter under review. The BNF $\rightarrow$ FPL syntax preprocessor, semantic/language translator, scanner, and FPL parser instruction interpreter were all completed and debugged to the extent that it was possible to produce a 1-pass compiler for a small subset of ALGOL and successfully parse source programs in that language. Extensive testing of the TWS was also provided by the Tranquil group and by a group of 20 students in CS109 who were assigned the task of specifying simple languages. It was found necessary to modify the syntax of Tranquil in several minor ways before the complete mapping of the BNF grammar into an FPL recognizer was accomplished by the preprocessor.

3.1.2 Syntax Preprocessor

Several minor bugs still have to be removed from the preprocessor. Some ideas for optimization of both the algorithm and the generated FPL statements are to be implemented. A feature to enable the system to convert a BNF grammar, which includes a restricted use of the empty production (i.e., $A \rightarrow \lambda$), has been implemented. Research efforts will be centered on error recovery and the possible use of limited top-down parsing when the present algorithm fails.

3.1.3 ISL and ISL Translator

During this quarter, the ISL Translator described in the previous report was finished and completely debugged. During this process, ISL (the Illinois Semantic Language) underwent several modifications: certain commands which had been tentatively included in the Language were dropped because they were found to be easily replaced by regular ALGOL commands. On the other hand, as ISL was used to describe the semantics of high-level languages, it was found necessary to specify new constructs which were then introduced into the semantic language. The present version of the ISL Translator receives as input the pass 1 semantics of a given language, written in ISL, and outputs an equivalent Burroughs ALGOL program. This program is then merged with a parser program (fixed for all languages) and the result is a pass 1 compiler for the given language. Besides tests with a small subset of ALGOL, the ISL Translator has been used to process a considerable part of the pass 1 semantics of the high-level language Tranquil.

The preparation of documentation for the ISL Translator as well as writing the final report containing the description of ISL is now under way.

Several small modifications will be introduced in the ISL Translator to make it "n-parse compatible". That is, the present translator has certain particularizations that make it capable of translating only semantics for pass 1. The modified translator will be able to translate semantics for pass n where n is any given interger, $n \geq 1$. Finally, it is our purpose to eliminate the "merge" operation described above in which the translator output is combined with the parser to yield a compiler. It is anticipated that in the final version, the ISL Translator, itself, given the pass-n semantics in ISL, will produce a pass-n compiler. Furthermore, given a set of n semantic programs for pass 1, pass 2, ..., pass n, the translator should be able to produce the complete n pass compiler.

An independent effort will be devoted to the preparation of a new ISL translator. This, rather than being a "tour-de-force" as the first one, is going to be obtained using the TWS system itself.

3.2 Tranquil

The syntax specification of the first version of Tranquil has been fixed and has been mapped by the TWS syntax preprocessor into an FPL parser. The major effort, during this quarter, has been the specification of the pass 1 semantics from which intermediate language code and descriptor tables are output. The pass 2 semantics, which will produce ILLIAC IV machine code, are now being specified. The virtual completion of the TWS project at the end of the next quarter should facilitate rapid progress towards completion of the Tranquil I compiler during the ensuing quarter. Some attention will also be given to the specification of Tranquil II.

3.3 Gleipnir

The syntax specification for Gleipnir has been completed and about 30% of the semantics for version I has been coded. This version will be a one-pass compiler which will include most of the features listed in the original language specification (ILLIAC IV Document No. 156)[8]. Work has been started on the specification of a high level macro facility which is expected to greatly increase the flexibility of the language.

3.4 CAT

3.4.1 Introduction

Over the last year, it has become evident that the departure from conventional machines introduced by the ILLIAC IV system requires that the general programmer be much more familiar with the internal structure of the machine than he has ever had to be before. In particular, efficient use of system parallelism requires that the programmer lay out his data on disk and in memory very carefully. However, most users will be unwilling to spend the time necessary to develop complex storage schemes and the resulting programs which use them. In this light,it is essential to develop suitable extensions to Tranquil to allow easy storage allocation and input-output.

3.4.2 CAT Compiler

In its highest conception, CAT (Tranquil II) will allow the user to describe the region over which his problem is to be solved in a symbolic notation. The user then uses this symbolic notation to provide control information (index sets) in the body of his program.

Furthermore, the user need not concern himself with core size or blocking. Storage allocation and disk blocking will be taken care of by the Tranquil II compiler in such a way that ILLIAC IV appears to be an infinite machine to the user.

The Tranquil II compiler will accept code written with ordinary Tranquil I statements, but the statements must be modified so that most index sets and loop controls are symbolic identifiers previously defined to represent physical regions and meshes with arrays of variables defined over them.

The output of the compiler will be ILLIAC IV machine code in three functional divisions: algorithmic kernels which represent the central core of the algorithm being used over the mesh of data, I/O instructions, and stencil loaders which provide control loops between the kernels and the I/O instructions. These stencil loaders are responsible for proper data buffering between various blocks of information on disk and in memory. Memory and disk allocation are also set up in the compiler.

The compiler itself will consist of three major parts (Figure 1). The geometry processor uses the geometry delcarations to define index sets over the regions used in the Tranquil I program and defines boundaries and other information about the mesh involved. The slicer uses the Tranquil code and the output of the geometry processor to order the calculation for efficient I/O utilization, interdata-block communication, and storage allocation--thereby slicing the Tranquil I code and the index sets produced by the geometry processor. Finally, the Tranquil I compiler produces machine code from the Tranquil I statement output of the slices.

3.4.3 Use Description of CAT

3.4.3.1 Introduction

A revised edition of an old ADI program was completed which was then used to show the various properties of the CAT language. There was also begun a study of several other numerical methods for

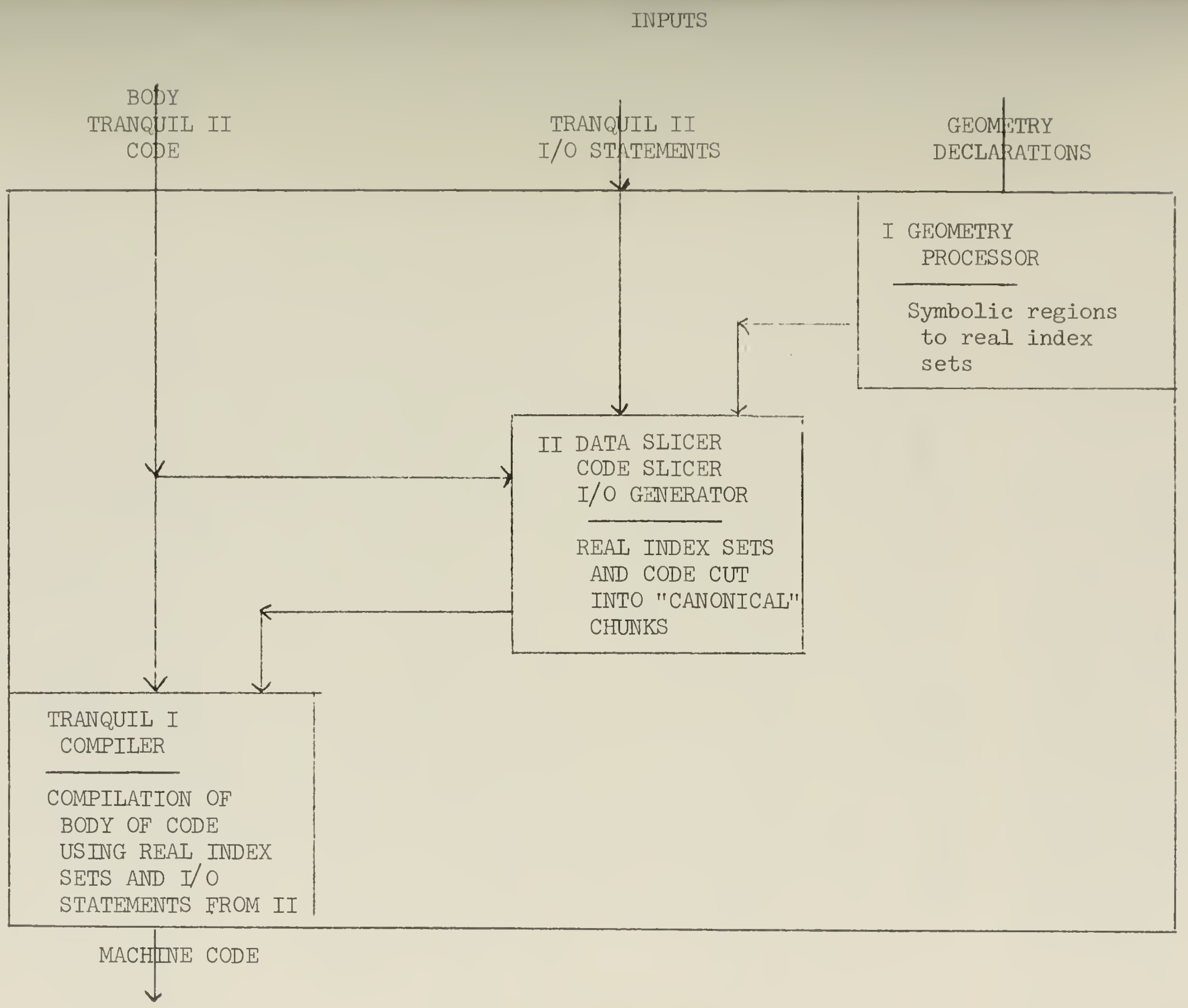

INPUTS
BODY TRANQUIL II CODE
TRANQUIL II I/O STATEMENTS
GEOMETRY DECLARATIONS
I GEOMETRY PROCESSOR
Symbolic regions to real index sets
II DATA SLICER CODE SLICER I/O GENERATOR
REAL INDEX SETS AND CODE CUT INTO "CANONICAL" CHUNKS
TRANQUIL I COMPILER
COMPILATION OF BODY OF CODE USING REAL INDEX SETS AND I/O STATEMENTS FROM II
MACHINE CODE

Figure 1. TRANQUIL II COMPILER

solving elliptic partial differential equations--namely: Gauss Siedel, point Successive Overrelaxation, and line SOR. The first two of these methods are in the process of being coded in Tranquil.

During this quarter some geometry statements were also developed that are to be added to Tranquil. A preliminary document on the syntax of the geometry specifications has been completed. The following presents the geometry ideas that it is believed should be available to ILLIAC IV users. This report is by no means exhaustive.

3.4.3.2 Sample Use of CAT

One of the goals of CAT is to facilitate the description of regions over which partial differential equations are solved. First, consider the description of plane curves--such as lines, circles, ellipses, hyperbolas, and others. The importance of the plane curves is that they will give the boundaries of the region in the plane that is of interest.

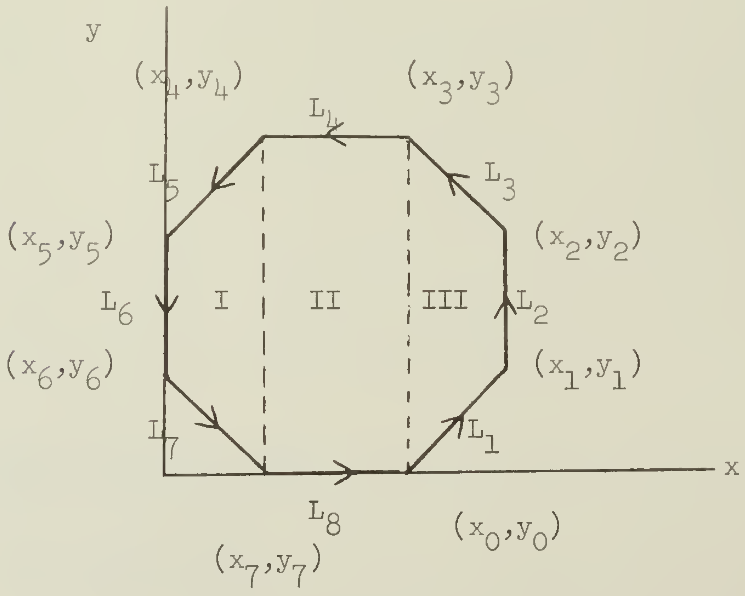


Figure 2

The following is an example. It is desired that a rectangular grid over the octagon shown in Figure 2 be used. It is apparent that which of the eight sides of the octagon y = I x H intersects as I varies from 1 to N (when N = y_4/H and H is the step size in the y direction) will want to be known. Also, the x coordinates of the intersection of the line y = I x H and the sides of the octagon will be needed. This knowledge can best be obtained by using procedures. So, the plane curves will first be declared by some declaration. For example:

```
PLANE CURVE L1, L2, ..., L4;
```

then the following will be set:

```
L1: = LINE ((X1, Y1), (X2, Y2));
L2: = PARABOLA (                );
L3: = ELLIPSE (                 );
L4: = HYPERBOLA (               );
```

such that each, L1, L2, L3, and L4, is completely defined. Explanatorily, by saying LINE, what kind of plane curve is wanted has been described to the computer. The contents of the parentheses completely describes a particular curve and also gives it an orientation. For example, in describing the line, the contents of the parentheses are the two end points of the line segment. Note that by taking the first end point as the tail and the second as the head, L1 can actually be thought of as a vector.

Once the plane curves and corresponding procedures are defined, it would then be desirable to define the three subregions of Figure 1. That is to say, there is wanted some way to define regions in the plane, given the plane curves. Suppose two plane curves, L5 and L6 as shown in Figure 3 are given, and the region bounded by these two curves is wanted.

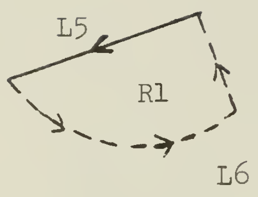


Figure 3

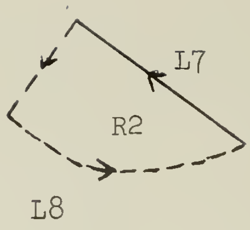


Figure 4

It is declared

```
REGION R1
R1 = L5 × L6;
```

Hence, R1 is the closed region shown in Figure 3; i.e., since L5 and L6 do not intersect, the head of L5 is connected to the tail of L6, and the head of L6 is connected to the tail of L5. If L7 and L8 do not intersect, as in Figure 4, then the head of L7 is connected to the tail of L8. Therefore, it gives the region R2.

Once the regions are defined, they must be able to have boolean operations performed on them; i.e.,

```
R3: = R1 ∪ R2;
R3: = R1 ∧ R2;
R3: = R1 ∪ R ∪∧ R4;
```

and others. In other words, it is wanted in some sense to treat these regions as sets.

Proceeding, the three subregions of Figure 2 can now be described.

```
REGION  RI, RII, RIII;
RI: = L7 × L5 × L6;
RII: = L4 × L8;
RIII: = L3 × L1 × L2;
```

The final major contribution that the geometric language should have is a thing called lattice. The lattice will be a collection of mesh points over a particular region. Therefore, there will be three input variables for a two-dimensional lattice. The first will be the region; the second will be one of the step sizes along with a curve which is subdivided; and the third will be another step size and the curve which is subdivided. In which case, let L1 be the x axis and L2 be the y axis, and let RI, RII, and RIII be the regions of Figure 2.

```
LATTICE LL1,LL2,LL3;
LL1:=LATTICE POINT (RI,L1:  H1,L2:K1);
LL2:=LATTICE POINT (RII,L1:  H2,L2:K2);
LL3:=LATTICE POINT (RIII,L1:  H3,L2:K3);
```

Hence, LL1, LL2, and LL3 can be thought of as sets of ordered pairs of indices such that if (I,J) is in LL1, then (I × H1, J × K1) is in RI. The reason for also giving a plane curve in the second and third coordinate to the input of the lattice is that it will allow for meshes other than rectangular. Once the lattices are defined, then LL1, LL2, and LL3 can be thought of as index sets; i.e.,

```
FOR (I,J) SIM (LL1) DO BEGIN
................
................
................
END;
```

Concluding this section on CAT, it should be pointed out that each of the topics discussed above should be able to be extended to more than two dimensions; i.e., the plane curves should be extended to volumes or regions in three dimensions, and lattices should be extended to sets of ordered triples.

3.5 SYSTEM K Group

3.5.1 Introduction

The work performed by this group during this subject period can be classified under the following headings:

1) B5500 software maintenance,
2) ILLIAC IV simulator and one-for-one assembler maintenance,
3) debugging system,
4) assembler system, and
5) operating system.

3.5.2 B5500 Software Maintenance

This group, as well as continuing its informal consultancy service on B5500 usage to the rest of the ILLIAC IV project, provided the following utility routines: 1) a log routine for machine-time cost itemization; and 2) disk library search, building, merging, and listing routines.

In addition to this work, "file security" has been installed on the B5500. This is to provide user identity and to prevent files being overwritten.

3.5.3 ILLIAC IV Simulator and One-for-One Assembler Maintenance

These programs have been in field test during this period, and sundry bugs have been removed. Extra work entailed by documentation changes in respect to ILLIAC IV design is under way. It is the responsibility of the SYSTEM K group to maintain the version of the Sankin timing simulator that the University now has.

3.5.4 Debugging System

A specification has been made which allows early use of this system. This specification is being formally described, and immediate implementation for use with B5500 ALGOL is envisaged.

3.5.5. Assembler System

A great deal of research work and detailed discussion has been put in on this project. A language form has been designed. The functions of the assembler and linking loader have been brought together under one logical viewpoint. It is hoped to issue a detailed technical description of the macro-assembler and linking loader features of this new software processor during the next quarter.

3.5.6 Operating System

ILLIAC IV Document No. 181[9] is an outline specification of a proposed operating system for ILLIAC IV/B6500 combination (q.v.). But, this document contains some omissions in the area of detailed technical specification. Work is proceeding to supply these deficiencies.

4. APPLICATIONS

4.1 Mathematical Applications

4.1.1 Introduction

This quarter saw the continuation of earlier work on problems in partial differential equations, matrix codes, and special functions calculations. Work also continued in the study of general programming techniques for ILLIAC IV. The emphasis shifted, in most instances, to simulation studies and debugging of assembly language codes. Work was initiated in two areas: the solution of systems of linear ordinary differential equations and polynomial root finding codes.

4.1.2 Differential Equations

4.1.2.1 Ordinary Differential Equation Solving on ILLIAC IV

Work is proceeding on solving a system of simultaneous differential equations by utilizing the inherent parallelism of ILLIAC IV. The particular equations under examination are of the form:

$$\frac{dX_k}{dt} = \sum_{i,j} f_{ij}^{(K)} X_i(t)X_j(t) + \sum_{i,j} g_{ij}^{(K)} X_i(t-\tau) +$$

$$\sum_i c_i^{(K)} X_i + a^{(t)}$$

One method, based on fourth order Runge-Kutta, for a system of 64 such equations has been coded. Because of the similarity of the equations, an increase of speed of a factor equal to the number of equations is theoretically possible over a serial computer. However, this involves solving each equation in separate Processing Elements, and routing is required in order to obtain all the values necessary for each PE to proceed to the next step. Therefore, optimization centers mainly on storing data and evaluating equations in such a way to minimize routing. Also, in this particular problem, $f_{ij}^{(K)}$ and $g_{ij}^{(K)}$ are sparse matrices, and some may be larger than others. This presents the problem of some PE's sitting idle waiting for the others to finish. It is planned to assemble and simulate this program on the Burroughs 5500 to test its feasibility and to further optimize it for ILLIAC IV.

4.1.2.2 <u>Alternating Direct Implicit Method</u>

During this quarter, work has been proceeding on an algorithm that uses the alternating direction implicit method to solve the partial differential equation:

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(K \frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(K \frac{\partial u}{\partial y}\right)$$

across a 64×64 rectangular mesh. The differencing equations used by this method are:

for rows (1) $-K_{i,j}\, u^{1/2}_{i-1,j} + [(K_{i,j} + K_{i+1,j}) + c]\, u^{1/2}_{i,j} -$

$K_{i+1,j}\, u^{1/2}_{i,j+1} = K_{i,j}\, u^{0}_{i,j-1} + [-(K_{i,j} + K_{i,j+1}) + c]$

$u^{0}_{i,j} + K_{i,j+1}\, u^{0}_{i,j+1}$

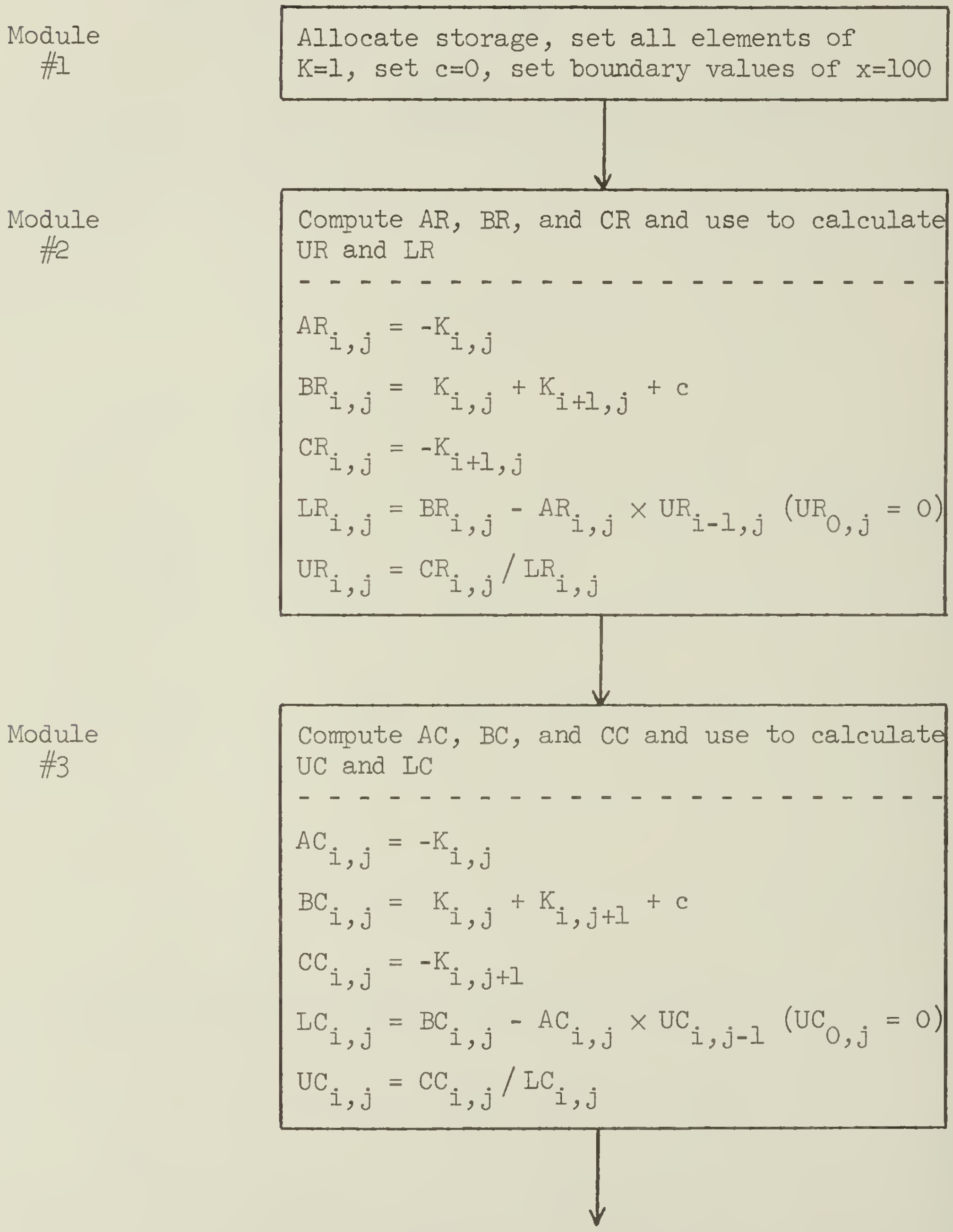
for columns (2) $-K_{i,j}\, u'_{i,j} + [(K_{i,j} + K_{i,j+1} + c]\, u'_{i,j} -$

$K_{i,j+1}\, u'_{i,j+1} = K_{i,j}\, u^{1/2}_{i-1,j} + [-(K_{i,j} + K_{i+1,j}) + c]$

$u^{1/2}_{i,j} + K_{i+1,j}\, u^{1/2}_{i+1,j}$

The specific case being studied is the heat equation, in which $K_{i,j} = 1$ for all elements of K and $c = 0$. The boundary values of u are taken to be 100.

This program has been assembled. For ease in debugging, it has been broken down into seven modules that are being simulated one at a time. At the same time, an ALGOL version of that module was written for a comparison of the answers.



Module #1

Allocate storage, set all elements of K=1, set c=0, set boundary values of x=100

Module #2

Compute AR, BR, and CR and use to calculate UR and LR

$AR_{i,j} = -K_{i,j}$

$BR_{i,j} = K_{i,j} + K_{i+1,j} + c$

$CR_{i,j} = -K_{i+1,j}$

$LR_{i,j} = BR_{i,j} - AR_{i,j} \times UR_{i-1,j} \quad (UR_{0,j} = 0)$

$UR_{i,j} = CR_{i,j} / LR_{i,j}$

Module #3

Compute AC, BC, and CC and use to calculate UC and LC

$AC_{i,j} = -K_{i,j}$

$BC_{i,j} = K_{i,j} + K_{i,j+1} + c$

$CC_{i,j} = -K_{i,j+1}$

$LC_{i,j} = BC_{i,j} - AC_{i,j} \times UC_{i,j-1} \quad (UC_{0,j} = 0)$

$UC_{i,j} = CC_{i,j} / LC_{i,j}$

Module #4

Calculate right-hand sides for rows

$$RHSR_{i,j} = K_{i,j}\ X_{i,j} + [-(K_{i,j} + K_{i,j+1}) + c]\ X_{i,j} + K_{i,j+1}\ X_{i,j+1}$$

Module #5

Compute $X^{1/2}$ for rows using forward and backward sweeps with Gaussian elimination

$$ZR_{i,j} = (RHSR_{i,j} - AR_{i,j} \times ZR_{i-1,j})/LR_{i,j}$$

$$(ZR_{0,j} = 0)$$

$$X^{1/2}_{i,j} = ZR_{i,j} - UR_{i,j} \times X^{1/2}_{i+1,j} \quad (X_{N+1,j} = 0)$$

Module #6

Calculate right-hand sides for columns

$$RHSC_{i,j} = K_{i,j}\ X_{i-1,j} + [-(K_{i,j} + K_{i+1,j}) + c]\ X_{i,j} + K_{i+1,j}\ X_{i+1,j}$$

Module #7

Compute X for columns as it was done for rows

$$ZC_{i,j} = (RHSC_{i,j} + AC_{i,j} \times ZC_{i,j-1})/LC_{i,j}$$

$$(ZC_{i,0} = 0)$$

$$X_{i,j} = ZC_{i,j} - UC_{i,j} \times X_{i,j+1} \quad (X_{i,N+1} = 0)$$

4.1.2.3 Hydrodynamic Code

Tranquil codes for the numerical solution of the Eulerian hydrodynamic equations in two-dimensional cartesian and cylindrical coordinates and three-dimensional cartesian coordinates are near completion. In two dimensions, the density, velocity components, and specific energy of the fluid are stored entirely in core in skewed fashion for a mesh which is no larger than 256×125. In three dimensions, again the hydrodynamic variables are stored skewed but $10 \times 10 \times 256$ blocks are processed at a time. This block requires about 1/3 of the available storage in core. The remaining core storage is reserved for I/O flow.

Another code being written utilizes "checkerboard" storage. This code was begun during this quarter and is nearing completion. The "checkerboard" storage scheme will allow an additional phase to be added to the code which will trace the path of "mass-less" particles. This will provide additional information to the user as to how the fluid is behaving.

4.1.2.4 Hockney's Method

Hockney's[10] method for solving Poisson's equation has been modified for the simulator and is now being tested. Results from a test case using the Gauss-Seidel method on Poisson's equation have been obtained on the Burroughs B5500. These results will be compared with the results from the simulator in order to test the simulator.

4.1.3 Matrix Problems

4.1.3.1 Algorithms and Error Analyses

Algorithms for Gauss Jordan and regular Gaussian elimination have been coded in assembly language for matrices of order n when $n \leq 256$. These have been coded so that the algorithms may be used as a subroutine to the partitioning algorithm for the inversion of arbitrarily large matrices.

Current work is being done in the following areas. Double precision routines in Gaussian and Gauss Jordan elimination are being worked on. Also, a routine is being coded for the iterative refinement of an approximate inverse obtained from one of the above algorithms. The algorithm is:

$$X_{k+1} = X_k + X_k(I-AX_k)$$

in which case A is the original matrix; X_k the approximate to the inverse; I is the identity matrix; and X_{k+1} is the new and hopefully better approximate to the inverse. The product AX_k must be computed in a higher precision than the rest of the calculations. The difference $I-AX_k$ must also be in a higher precision than the main body of computation.

This algorithm will converge only if the following inequality is satisfied. $1.01\ (n^3 + 3n^2)\ \rho\ \mu\ ||A||_\infty\ ||A^{-1}||_\infty < \frac{1}{2}$, where $||A||_\infty$ $||A^{-1}||_\infty$ is the condition number of the matrix A; n is the order of A; μ is 2^{1-t} where t is the number of bits of mantissa (48 for ILLIAC IV); and ρ is approximately 1.

4.1.3.2 <u>Inversion by Partitioning</u>

A Tranquil code was written to invert a matrix by partitioning. An algorithm partitions a matrix S and its inverse as follows:

$$S = \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} \qquad S^{-1} = \begin{bmatrix} K_1 & L_1 \\ M_1 & N_1 \end{bmatrix}$$

$$D_{i-1} = \begin{bmatrix} A_i & B_i \\ C_i & D_i \end{bmatrix} \qquad N_{i-1} = \begin{bmatrix} K_i & L_i \\ M_i & N_i \end{bmatrix} \quad i=2,3,\ldots,r$$

The partitioning takes place when

$$A_i \text{ and } K_i \text{ are } 256 \times 256, \text{ and}$$
$$D_r \text{ and } N_r \text{ are } p \times p \text{ -- when } \leq 256.$$

By using the following formulas,

$$K_i = (A_i, B_i D_i^{-1} C_i)^{-1}$$
$$L_i = K_i B_i D_i^{-1}$$
$$M_i = -D_i^{-1} C_i K_i$$
$$N_i = D_i^{-1} - D_i^{-1} C_i L_i$$

D_r can be inverted and D_{r-1} can be calculated. Then, D_{r-2} is calculated. This is continued until K_1, L_1, M_1, and N_1 are obtained. At each step $A_i - B_i D_i^{-1} C_i$ must be inverted. This is necessary because of the way S was partitioned, and $(A_i - B_i D_i^{-1} C_i)$ is 256×256 for all i. The inversion, therefore, can be done in core.

The current work is focused on multiple precision arithmetic and on iterative refinement of matrix inversion.

4.1.3.3 Sparse Matrix Inversion

A storage scheme for non-zero elements and algorithms using Gauss-Jordan Elimination is almost determined. It will be coded by using ILLIAC IV assembler language.

The following things must be considered: 1) reduction of wasted memory, and 2) program efficiency. The first step will be to code the storage scheme in the most simple form without taking account of 1 and 2 mentioned above.

Since only non-zero elements are stored in the memory, the following things must also be considered: (a) pivot search, (b) how to get a complete row or column, and (c) change of the number of non-zero elements during processing. The accomplishment of (a) and (b) is not too difficult, but (c) will be the key point for the sparse matrix inversion. Therefore, future effort will be concentrated on (c).

4.1.4 Math Subroutines

4.1.4.1 Root Finding

This quarter has seen the initiation of an effort to develop a comprehensive polynomial root finding package for ILLIAC IV. The first algorithm developed finds simple roots by multisectioning a given range, examining each subdivision for the presence of a root, and repeating this process on these subdivisions containing a root. After the simple roots are obtained, the polynomial is deflated, and a Lynn-Bairstow procedure, which attempts to factor quadratic terms from the deflated polynomial, is used to find all multiple roots.

A Tranquil code for the Lynn-Bairstow method is completed. Meanwhile, the code for the multisectioning is nearing completion. In addition, Tranquil codes for standard algebraic and trigonometric functions and routines are being written to test the efficiency of the assembly language codes for these subroutines.

4.1.4.2 Special Functions Library

4.1.4.2.1 Introduction

During this quarter, work has continued on the increasing of ILLIAC IV's Special Functions subroutine library. Codes have been written in assembly language for a 32 bit floating point, square root routine and for 64 bit floating point, sine, cosine, and exponential routines.

4.1.4.2.2 Square Root

Let Z denote a positive floating point number whose square root is to be computed. Let $Z = 2^{2k} \cdot m$ where 2k is the exponent of the floating point number and $1/4 \leq m < 1$ is the mantissa. To compute $\sqrt{Z}$, Newton's method will be used:

$$y_{n+1} = 1/2\ (y_n + Z/y_n); \quad n = 0, 1, 2, \dots$$

for which a close starting value y_0 is needed.

Since $\sqrt{Z} = 2^k \cdot \sqrt{m}$, an approximation to $\sqrt{Z}$ may be obtained from an approximation, A, to $\sqrt{m}$ by taking $y_0 = 2^k \cdot A$. In this way, a suitable starting value y_0 for Newton's method by approximating $\sqrt{m}$ for $1/4 \leq m < 1$ is found. $A = .34314575 + .68629150\ m$ yields a close enough approximation to $\sqrt{m}$ such that two iterations of Newton's method will yield the necessary seven digits of accuracy. Based upon the known time for each instruction involved, the entire program takes approximately 13 microseconds.

4.1.4.2.3 Sine and Cosine

Let x be a floating point number whose sine and cosine are to be computed. SINx and COSx are computed by the Chebyshev polynomial approximation to SIN$1/2\pi x$ and COS$1/2\pi x$, respectively, where $-1 \leq x \leq 1$.

In general, if $x = 2N + t$ where N is an integer and $-1 \leq t \leq 1$, then SIN$1/2\pi x = (-1)^N$ SIN$1/2\pi t$ and COS$1/2\pi x = (-1)^N$ COS$1/2\pi t$. This way the range of x is expanded to $-\infty < x < \infty$.

4.1.4.2.4 Exponential

To calculate e^x in floating binary form for any real x, let $x \log_2 e = N-t$ where N is an integer and $0 \leq t \leq 1$. Then $e^x = 2^N . 2^{-t}$. Then the approximation of 2^{-t}, in which $0 \leq t \leq 1$, can be made by a Chebyshev polynomial expansion.

4.1.5 Matrices

4.1.5.1 Eigenvalue Problems

During this quarter, work was done in a number of areas. Investigation of eigenvalue problems for matrices of sizes larger than the immediate storage access on the ILLIAC IV took place. Moreover, general kernels for the Jacobi's method were formulated, and problems with stencil operations were discussed. The completion of a flow chart showing the sequence of the computations involved also was accomplished. An eigenvalue report is forthcoming.

4.2 Linear Programming

4.2.1 Introduction

This quarter has been a period during which the long-range aims of the linear programming group have been defined and progress has been made in the investigation of algorithms suitable for ILLIAC IV implementation.

4.2.2 Definition of Aims

It has been decided that at least two types of linear programming codes should be written. For "small" problems (less than 4000 rows) the algorithm used will probably be the product-form of the revised simplex method. For large problems (up to, perhaps, 20,000 rows) it will be desirable to use an algorithm involving the minimal number of iterative steps possible, so as to reduce the likelihood of round-off errors which may well lead to meaningless solutions. To this end, a single-pass, non-iterative solution procedure based on the properties of convex polyhedral cones is currently under examination. Investigations are also being made of several methods of improving the computational efficiency of the simplex method. The results of these investigations will appear in the next Quarterly Progress Report.

4.2.3 Revised Simplex Algorithm: Explicit Inverse

A small (5×10) linear programming problem was formulated for testing the LP code which appeared in ILLIAC IV Document No. 171[11]. The code for problem data generation has been written and symbolic addresses in the PEM, CU, and DB are properly assigned for each entry. The code will be tested on the ILLIAC IV simulator in the B5500, with the aim of testing the program logic and assessing the working efficiency of the algorithm under the parallel processing format of ILLIAC IV.

4.2.4 Revised Simplex Algorithm: Product-Form

The product-form algorithm has been carefully examined and a multiple-pricing scheme has been developed to take advantage of the parallel design of ILLIAC IV. For problems of up to 4,000 rows, this method appears likely to result in very large speed advantages. However, data input problems have arisen and means are presently being sought to make the solution speed less I/O bound. Another source of concern is the likelihood of serious rounding problems with matrices with dimensions of the order of 4,000 rows. These problems are currently under consideration. Meanwhile, flow-charting of the algorithm is proceeding, preparatory to coding for tests on the simulator.

4.2.5 Future Plans

Progress during this quarter has been such that it is expected that the evaluation and testing of alternative algorithms can be completed during the next quarter; so that the coding of selected algorithms for implementation in ILLIAC IV can be started.

4.3 Graphics

More detailed debugging of the previously described graphical display system was done. The system has been extensively tested by class use and appears to be debugged. A second version of the system incorporating many minor improvements is being tested.

During the debugging process, a change in the common declaration statements was required. This, in turn, afforded the opportunity to insert frequently used constants--such as pi and degree to radian conversion constant--in common.

Also, the off-axis rotation routines were changed; so that, rather than testing for zero, they tested for variable EPS--a small number representing a system zero-- to compensate for possible round-off error.

4.4 Fingerprint Pattern Studies

4.4.1 Introduction

The parallel processing capabilities of ILLIAC IV should be particularly applicable to the class of pattern processing procedures that perform a given operation of many different points of a pattern. With this thought in mind, a group was formed to investigate the processing steps used in the manipulation of non-numeric information--specifically pattern or picture information.

Rather than attempting to deduce a set of useful procedures by considering the broad specturm of pattern processing applications, the effort will concentrate on the properties of a pattern of immediate practical significance with a well defined, regular structure, namely fingerprint patterns. Fingerprint dimensions are well known and do not vary widely. An extensive scheme of fingerprint classification has already been applied to very large fingerprint files. Although the present classification scheme, the Henry classification system, may not be applicable to machine extracted pattern data, it will provide a starting point and a standard of comparison.

4.4.2 Fingerprint Recognition

A hand-digitized fingerprint was obtained for testing initial schemes. This fingerprint was coded on a 64 x 64 grid, and it was noted that a higher grid density is desirable (probably 256 x 256).

An attempt was made to remove noise by following ridges and by reducing them to one grid interval thickness. This attempt failed because the ridge following scheme used only the previous direction of a ridge to determine the future direction and could not follow ridges through noisy areas. Partial success was obtained in determining ridge direction throughout the print by counting ridges in four directions from a given point. It is hoped that with a 256 x 256 grid many more directions can be used in determining ridge direction.

While ridge direction about a given point is being determined, a confidence level for the direction found can be computed. When directions and confidence levels have been calculated throughout the grid, these values can be smoothed by averaging the directions weighed by their confidence levels throughout an area. Noisy regions can then be found by comparing the initial directions and confidence levels with the smoothed versions. The noisy regions can then be searched for cores, deltas, and minutiae.

4.5 Signal Processing

4.5.1 Applications of ILLIAC IV to Phased Array Radar for Area Defense

The data processing requirements of the phased array radar system required for ICBM defense and tracking of the expected numbers of satellites in space cannot be handled by present, general purpose computers. This data is collected in parallel, and the phased array radars are parallel devices capable of collecting data from several different objects and different directions in parallel. Therefore, these radars require data processing equipment which can process this data in parallel.

During this quarter, the applicability of ILLIAC IV to the handling of the data processing requirements for large phased array radars has been under investigation. A report describing how ILLIAC IV can handle several of the major functions has been published[12]. These functions have been programmed for ILLIAC IV and run on the ILLIAC IV simulator.

Presently some of these functions are going through a revision to incorporate more complex and realistic techniques. The first program, which used least square techniques for the tracking of objects, is presently being changed to use Kalman Filter techniques for tracking. This new tracking algorithm will place more stringent requirements upon the computer both in processing time and in memory amount; however, an approach similar to the Kalman tracking is required in the real world to handle the tracking of objects.

Jointly, with Auerbach, the discrimination requirements placed upon the computer will be factored into the overall processing requirements. Investigation into what type of an operating system is required to be able to interleave all the different functions and to keep the computer system operating efficiently is taking place. The availability of the data processing equipment is required 100 percent of the time; therefore, the effects of hardware failure in ILLIAC IV and the ability to maintain the system while operating is being investigated.

4.6 ILLIAC IV Educational Program

An educational program was established during this quarter. The goal of this program is to furnish a method by which users of ILLIAC IV can become acquainted with the unique aspects of this computer. This educational program will increase the availability of the computer to more users. Users will include university persons and government (ARPA Network) persons. To fulfill this goal, formal course literature will be prepared which will contain the necessary information to give users knowledge of and accessibility to ILLIAC IV.

Also, during this quarter, the ILLIAC IV Programming Manual was delivered by Burroughs. The Manual in general explains ILLIAC IV programming; however, it does contain errors and some sections need clarification. A list of all recipients of the Manual has been maintained; so that, as they send comments and changes to the ILLIAC IV office and as these are reviewed and sent on to Burroughs, they can receive revisions and corrections and maintain a current Manual.

Some of the areas of clarification are syntax rules for coding each field of the CU and PE instruction, assembler pseudo-operations, and present I/O arrangements for the assembler. A liaison is being maintained to have changes incorporated in the Manual and printed and distributed.

REFERENCES

1. Heimerdinger, W., and Sterling, W. A Program for Generating a Set of Paths for Fault Location GRAPH/MARK 10. (February 6, 1968), ILLIAC IV Document No. 172.

2. Kato, M. List of Diagnostic Memorandums. (March 8, 1968), ILLIAC IV Document No. 178.

3. Kato, M., Koga, Y., and Naemura, K. Diagnostic Test Patterns and Sequences for ILLIAC IV PE. (February 8, 1968), to be published as ILLIAC IV Document.

4. Kato, M., Koga, Y., and Naemura, K. Preliminary List of Diagnostic Dictionaries for ILLIAC IV PE. (February 23, 1968), ILLIAC IV Document No. 175.

5. Koga, Yoskiaki. Diagnosis of Carry-Save Adder, Multiply Decoder Gates and Multiplicant Select Gates with PEX and Multiply Operation. (January 3, 1968), ILLIAC IV Document No. 170.

6. Ulrich, E. G. "Time-Sequenced Logical Simulation Based on Circuit Delay and Selective Tracing of Active Network Paths." 1965 ACM National Convention, pp. 437-448.

7. Reiss, R. F. "The Digital Simulation of Neuro-Muscular Organisms." Behavioral Science, October 1960, pp. 343-358.

8. Lawrie, Duncan. GLEIPNIR: A General Purpose Low Level List Processing Language; I. Preliminary Specifications. (September 28, 1967), ILLIAC IV Document No. 156.

9. Saville, Nicholas. A Description of the System Specification of OSK: An Operating System for ILLIAC IV. (April 8, 1968), ILLIAC IV Document No. 181.

10. LaFrance, Jacques. Implementation of a Fast Direct Solution of Poisson's Equation Using Fourier Analysis on ILLIAC IV. (November 29, 1967), ILLIAC IV Document No. 166.

11. Chen, Frank. Linear Programming Implementation in ILLIAC IV; I. Revised Simplex Method. (January 12, 1968), ILLIAC IV Document No. 171.

12. Knapp, M., Ackins, G., and Thomas, J. Application of ILLIAC IV to Urban Defense Radar Problem. (February 21, 1968), ILLIAC IV Document No. 173.